Published by Viglen Ltd.
Unit 7, Trumpers Way, Hanwell, London W7 2QA.
Tel. 01843 9903

The Floppy Disc Manual
For BBC Microcomputer Users

Contents

Appendices

1. Preface

The sections of this manual have been laid out in the order that they are likely to be needed.

This manual is written for 5¼ inch and 3 inch floppy disc systems and in most cases the treatment is identical. Where differences arise, these will be specifically mentioned.

Chapter 2 is a general description of disc drives. It can safely be left out if you are already familiar with disc drives.

Chapter 3 is a description of how to fit the disc interface to the computer. A disc system is far more *'intelligent'* than the cassette interface, and this *'intelligence'* is achieved by the disc interface which is situated within the computer rather than within the drives.

Chapter 4 covers connecting everything up together for the first time.

Chapters 5 to 9 explain the various commands which are available. These have not been put into alphabetical order, instead they have been grouped according to their function.

Chapter 5 covers formatting and verifying. Unless you have some formatted discs this will be the first thing you need to learn about.

Chapter 6 explains the simplest commands, i.e. those you will have already met on the tape system. These are all you shall need to get started.

Chapter 7 covers all the commands concerned with drive addressing, directories and libraries.

Chapter 8 covers the commands concerned with moving files about, like COPY, RENAME and DELETE.

Chapter 9 covers commands not falling into any of these categories, and the auto-boot facility.

Chapter 10 covers the use of files in BASIC using BASIC commands like PRINT#, INPUT#, EOF#, PTR# etc.

Chapter 11 covers file management on the disc, and how to avoid the dreaded *'can't extend'* message.

Chapter 12 is about memory management in the computer, why PAGE has suddenly changed to &1900, HIMEM, LOMEM, and *'No room'*. This section also contains hints on converting tape programs to run from the disc; possibly the most difficult thing to do of all.

Chapter 13 lists all the disc filing system commands along with the syntax and the abbreviated version of the command.

Chapter 14 lists the error numbers obtained when using the disc filing system commands in a BASIC program.

Appendices cover Care of Discs, Disc Sector Access, Drive Select Switchblock and Catalogue Sector Format.

Notes:-
Instructions to be typed in are either given a separate line, or enclosed in single quotes ('). The single quotes are never typed in themselves. Angle brackets (< or >) either mean the name of a key, like <caps lock>, or something general like the number of a drive e.g. <drv>. If something is enclosed in ordinary round brackets, it is optional. As an example, take:-

```
*CAT (<drv>)
```

This means that any of the following are legal:-

```
*CAT
*CAT 0
*CAT 1
*CAT 2
*CAT 3
```

The term *'file'* is just a general name for anything you can store on a disc, be it program, data, text, etc.

Many keywords can be abbreviated but they are shown in full in this manual to avoid confusion.

2. Introduction to disc drives

This section describes the nature of disc drives, and how their use differs from that of cassettes for the storage of programs and data. If you are already familiar with the use of disc drives, you can omit this section.

The use of discs instead of cassettes

The very fact that you have come to VIGLEN to buy a disc system shows that you already realise the limitations of a cassette tape recorder for the storage of programs and data. There are two main features of a disc system which make it superior to a tape system.

Firstly, the disc system can store and retrieve files at a much faster rate. This is because the cassette recording system has to convert the program to sound in order to store it on an audio cassette, limiting the rate of storage and retrieval by the frequency of sound. However, the disc recording system can record signals of a much higher frequency than sound, enabling the data to be stored and retrieved much more quickly.

Secondly, the disc system will look after the location of your files on the disc. When programs are stored on cassettes you need to keep track of their location yourself, probably using the tape counter and paper records. If you want to save a program between two others on the tape, you have to ensure that there is adequate room for it to prevent the second one being overwritten. When you upgrade to the disc system your computer is fitted with a Disc Filing System (DFS), which is a program that does all this *'housekeeping'* for you. It keeps a record on the disc, called the catalogue, of the files that are stored on it, their location and their length. The catalogue is always stored in the same place on each disc. When a file is required, the catalogue is loaded from the disc and used to find the location of the file on the disc. The DFS program then moves to the file and loads it without the need of manual intervention. Pressing a PLAY button is not required because a disc drive has no user operated controls. Movement is achieved by motors and a solenoid under the control of the DFS program in the computer.

Your disc system replaces the cassette system, however you can leave it connected if you want to. Instead of the tape recorder you have disc drives, and instead of audio cassettes you use discs. Blank discs can be bought just as you bought blank cassettes, but they must be formatted before use as described in *Chapter 5*. The disc drives plug into a socket underneath the computer, and operate through the 'floppy disc interface', without which the disc drives will not work. To use the disc drives, you put the disc into the drive and close the door; you can then LOAD and SAVE programs with exactly the same commands that you used with your tape recorder.

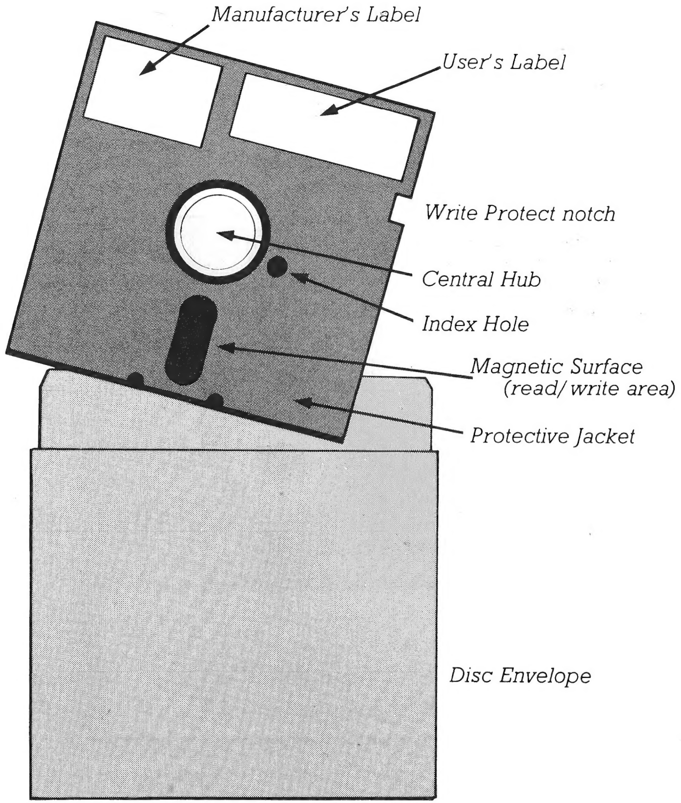
How discs work

Like cassette tape, discs are covered in a magnetic material but it is of higher quality to store data more densely. This is similar to having finer grain on photographic film.

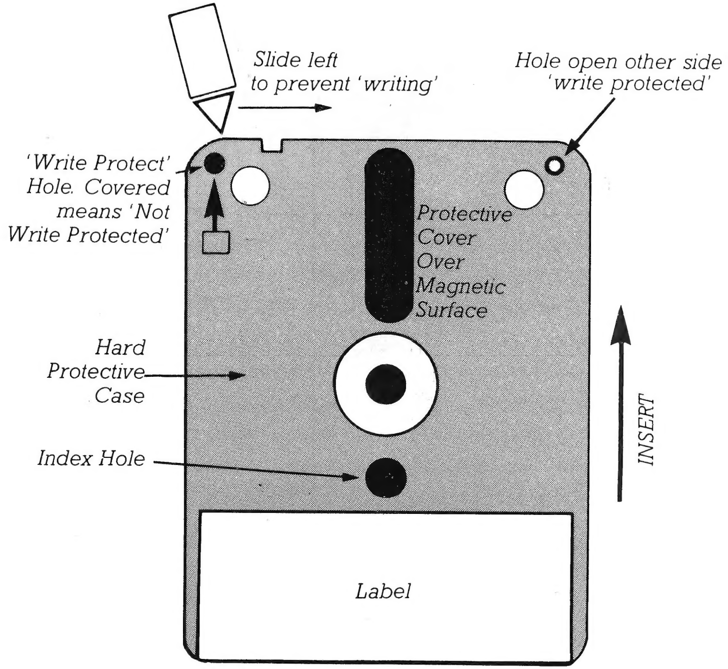
5¼ inch discs

5¼ inch floppy discs come in a plastic sleeve to protect them against dust and greasy fingerprints, the sleeves themselves being protected by a cardboard or paper sleeve.

An elongated cutout in the plastic shows where the read/write (i.e. playback/record) head can come into contact with the disc. As with the tape in a cassette, it is important that nothing touches the exposed surface of the disc. The small hole in the disc near the centre allows the disc drive to shine a beam of light through while the disc is rotating. A photocell on the other side of the disc gives a synchronising signal as the hole goes past, which is used to tell the DFS where the disc is.



5¼ inch floppy disc features



Slide left to prevent 'writing'
Hole open other side 'write protected'
'Write Protect' Hole. Covered means 'Not Write Protected'
Protective Cover Over Magnetic Surface
Hard Protective Case
Index Hole
INSERT
Label

3 inch compact floppy disc features

The notch on the edge of the disc is used like the tab on the back of a cassette, that is to prevent accidental erasure of programs. An opaque label (supplied with a box of discs) must be fixed over the notch to prevent write operations onto the disc. The discs can only be inserted in the drive in one direction.

3 inch discs

3 inch floppy discs are basically a smaller version of the 5¼ inch discs. The disc itself is protected by a hard plastic case and is totally covered so that you cannot actually touch any part of the floppy disc. You can use both sides of a 3 inch disc by flipping it over. The colour of the light on the front of the drive will change depending on the side that has been inserted. To prevent any data being written onto the disc there is a special red tab which should be slid to one side (using the tip of a ball point pen or similar) thus opening the small hole.

Operation of the drive

When the drive is operating the disc rotates at 300 rpm, and the read/write head records programs on concentric rings around the disc, which are called tracks. The head stays still while recording data, but a special motor called a stepper motor may move the head in and out to change between tracks. The stepper motor is so called because, unlike an ordinary motor, it turns in discrete steps. The mechanism of the stepper motor action differs in different makes of drives. In its most basic form the stepper motor turns a rod with a screw thread on it to which the head is fixed. As the rod turns, so the head moves either towards the centre or the outside of the disc. If you have a 40 track disc then the head will travel over the useable area of the disc in 40 steps of the stepper motor, one for each track. If you have an 80 track disc then the head will scan the same area in 80 steps.

The mechanism for 80 track discs has to be more accurate and the head has to be of a higher quality because it only has half the disc space to record the same data in, which is why 80 track disc drives cost more than 40 track ones. Naturally, an 80 track disc will store twice as much data as a 40 track one because the amount of data that can be stored on each track is the same.

If you have one of the drives which are switchable between 40 and 80 tracks then it is basically an 80 track drive, with a circuit which causes the stepper motor to count two steps at a time if the 40 track mode is selected.

Each track is divided up into 10 sectors, each consisting of 256 bytes (also known as a quarter of a 'K') so a 40 track machine stores 40*10*0.25K=100K bytes of data. Similarly, an 80 track machine stores 80*10*0.25K=200K bytes. In each case two sectors (i.e. half a 'K') are used by the catalogue.

If you have a double sided drive then a second head reads the other side of the disc, doubling the amount of storage space available. This may lead you to think that if you only have a single sided drive you can turn the disc over and use the other side, like a cassette. Unfortunately this is not possible for 5¼ inch discs because the synchronisation hole and the write enable notch will be in the wrong place. 3 inch discs may be turned over in order to use the other side.

When discs are manufactured, they are all made in the same way. They are then tested and graded according to how good they are, as single or double sided, single or double density. You can always use a disc which has been graded as better than you need, although you will pay for it. If you use a disc which has been graded as worse than your requirements then you may end up getting read errors and regretting it.

It is a common fallacy to think that 80 tracks is the same as double density. In fact, double density is a system of storing more data ON EACH TRACK. 80 track is commonly known as double track density.

VIGLEN drives are capable of working in double density mode though you must have a suitable double density controller and filing system to go with it. The ACORN, CUC and similar filing systems are only capable of single density usage.

3. Fitting the 'disc interface'

This chapter contains instructions to enable you to add the disc interface to a model 'B' BBC microcomputer which has the 1.20 operating system. If your BBC micro is already fitted with a disc filing system then you should skip this chapter.

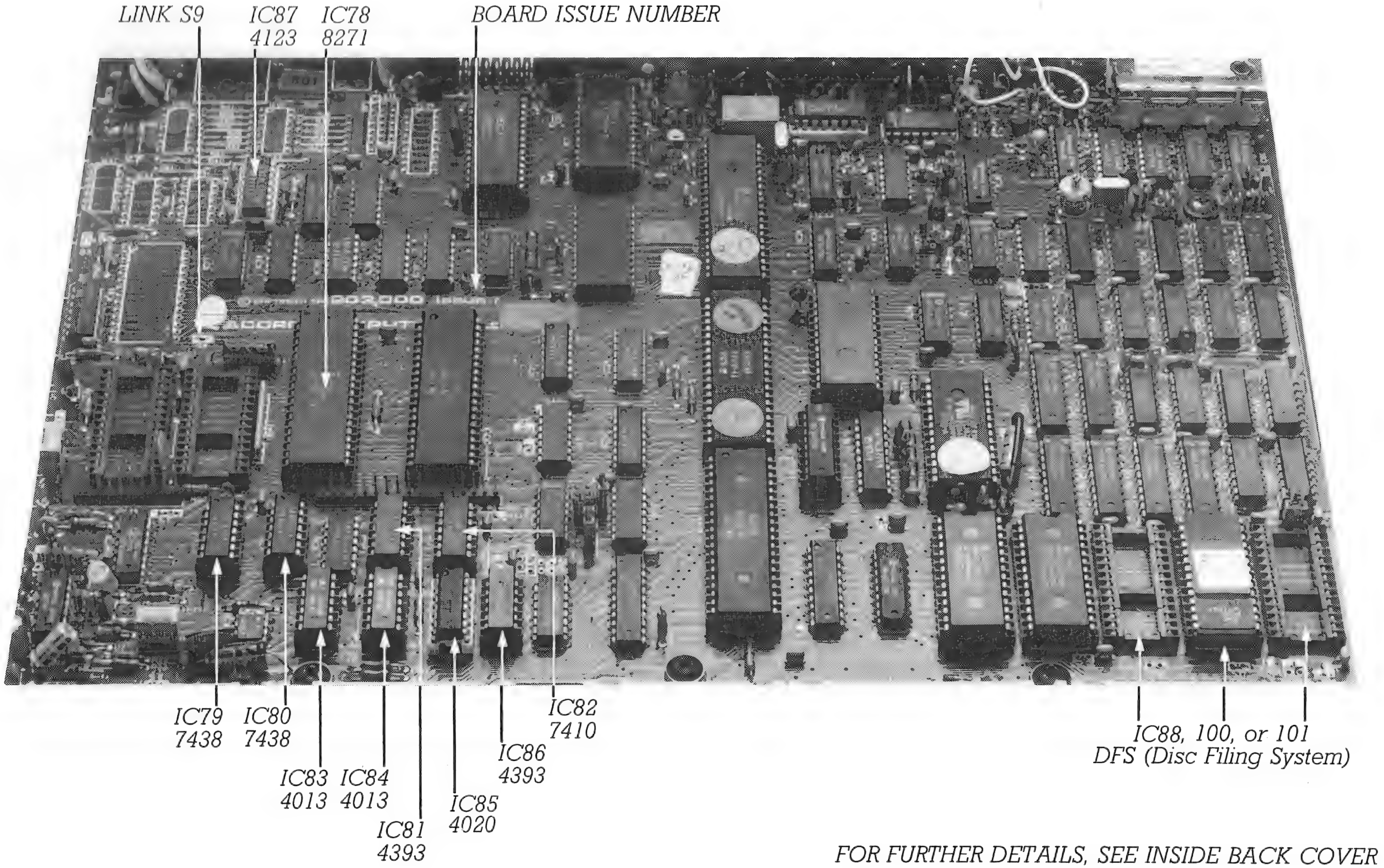
If you are unsure of the specification of your computer, you can find out by following these instructions:–

1. Switch the computer on. If it says *'BBC computer 16K'* then you only have a model A and you must first fit the extra 16K memory as well as some of the other model A to model B upgrade chips. If it says *'Acorn DFS'* or *'C.U.C. DOS'* or something similar then your computer has already been fitted with the disc interface.
2. Type '*FX0'. If the version number given for the operating system is less than 1.00 then you must fit your computer with the new operating system.
3. Look under the computer at the connector labelled *'disc drive'*. If the actual connector is not there then you will have to fit this to the computer as well.

Deciding whether to fit the interface yourself

Only attempt fitting the disc upgrade yourself if you are confident that you know what you are doing. The procedure is not particularly difficult but a mistake could be expensive!

Issue 4 boards onwards only need the ICs to be fitted and links to be changed. Previous issues of boards also need a track to be cut and some fiddly soldering done. The issue number of the board is screen-printed on the main board. If you have the older black linear power supply, visible where the mains lead goes into the computer, then you must use disc drives that have their own power supply. If your disc drives have a mains plug on the end of the power lead then you need not worry which power supply is in the computer as it does not have to supply the drives. The switch-mode power supply which is a dull silver colour may be able to supply drives if the power requirement is low enough. If you do have the older linear power supply then find out from your dealer whether the drives you intend getting, or have already bought, will run from the switch-mode power supply if you had one. This is certainly true of drives bought from VIGLEN. If this is the case then your best bet may be to get the switch-mode power supply fitted instead of paying out more for a self powered disc drive.



FOR FURTHER DETAILS, SEE INSIDE BACK COVER

Fitting procedure

1. Make sure the computer is unplugged from the mains supply before starting. Check that you have all the ICs according to the following list:–

I.C. No.	Type of I.C.
IC78	8271
IC79	7438
IC80	7438
IC81	4393 *or* 74LS393
IC82	7410 *or* 74LS10
IC83	4013
IC84	4013
IC85	4020
IC86	4393 *or* 74LS393
IC87	4123
IC101	2764

The numbers above are only the *'generic type'* of the IC. Manufacturers add their own letters and numbers to identify their own products so, for example, IC83 may be marked MC14013BCL. Don't get confused with the data code which is also often marked on the IC as the year and the week. The number '8123' implies the IC was made in the 23rd week of 1981.

2. Open the case of the computer by removing the four screws marked *'fix'*, there are two underneath and two at the rear. Remove the keyboard connector on the main board end as most of the new ICs go just below this connector. Ease the connector up evenly on both ends, otherwise the end pins will be bent as the plug comes free. Remove the keyboard by unscrewing the bolts fixing it to the case. Finally, unplug the flying lead to the speaker allowing the keyboard to be placed to one side.

3. If your board is previous to Issue 4 then perform the following modifications:– *DO NOT perform these modifications for Issue 4 boards and onwards!*
   a) Cut pin 9 of IC27 as near to the board as you can, then carefully bend it up a little.
   b) Cut the track that goes from that pin to the right hand side of link S9.
   c) Solder a short piece of plastic covered wire from the bent up pin 9 of IC27 to the right hand side of S9. Although this modification does not appear to make any difference to the circuit, it does in fact disconnect a track underneath IC27 which cannot itself be cut as it is completely hidden by the IC body.
   d) If your board is previous to issue 3 then solder a piece of wire across link S8.

4. Beware of static electricity while fitting the ICs. The CMOS ones, whose numbers begin with 4, are particularly sensitive. If you pick up an IC and walk around on a nylon carpet you will accumulate an electrical charge. As you plug the IC into the board the charge will flow through the chip, damaging it internally. To avoid this risk, touch the IC packaging and the tracks on the board at the same time. This will ensure that you and the ICs and the board

are at the same potential. If you should walk away then repeat this when you return. Also, don't wear a nylon shirt.

Now fit all the ICs except IC101, taking care to insert them the right way round. *Refer to the photograph for help.* Pin 1 is marked with a pip or spot and must be in the top left as you look at the board, pointing towards the power supply. Check by comparison with all the other ICs on the board, all but two are oriented the same way. It is useful to bend the pins against the table until they are parallel to help them go in easily. They always come with their pins bent out a little to fit industrial automatic component insertion machines.

5. Although the 2764 has been referred to as IC101 it can go in any of the four right hand ROM sockets. When the computer is first turned on, or <ctrl-break> is pressed, the operating system calls the ROMs in turn from right to left. This continues until a language ROM is found. Word processor ROMs count as language ROMs. If the operating system cannot find a language ROM then it complains by saying *'language?'*. If the 2764 DFS chip is in any one of the ROM sockets then the BASIC interpreter will switch onto a disc system. In order to get back to the cassette filing system you must enter *TAPE.

   Do not touch the leftmost of the five ROMs as this contains the operating system and must be in IC51. The most popular combination will be to fit the DFS in the rightmost position. Moving the BASIC ROM from IC52 to IC100 means that any extension ROMs you should buy later like FORTH or VIEW can just be plugged into IC52 and IC53 and BASIC will still be invoked when you switch on. Moving the BASIC ROM is optional.

6. Now set the links according to the following list. Don't worry if any of the links are already correct. The links may be either tiny loops of wire or little plugs that go over two out of three pins. They are labelled by the printing on the board in the same way as the IC positions are. North means away from you towards the back of the computer. Most links are three pins sticking up out of the board, with a plug over two of them. If, on for example S19, the two on the left are connected then pull the plug off and fit it over the two on the right. S21 is special, it is four pins with two plugs which can go east-west and east-west or north-south and north-south. All four pins are used in both cases.

S9	– remove link if present
S12	– remove link if present
S13	– remove link if present
S18	– North
S19	– East
S20	– North
S21	– East-West, East-West
S22	– North
S32	– West
S33	– West

Now refit the keyboard, with the loudspeaker lead and keyboard connector, and re-assemble the case.

When you turn on you will see:-

```
BBC Computer 32K
<disc rom id.>
BASIC
>
```

where <disc rom id.> may be 'Acorn DFS' or 'C.U.C. DOS' or something similar depending on the disc operating system EPROM (2764 chip).

If you now enter *HELP DFS you will obtain on the screen a list and syntax of the commands which you can use with your disc filing system.

4. Connecting the drives to your computer

This chapter describes how to connect your drives and get started.

Connecting the cables

There are two cables coming from the disc drives, one for the power required by the drives and one for the transfer of data in either direction. The latter is a flat cable called a ribbon cable which plugs into a socket underneath the computer labelled *'disc drive'*. The plug is inserted so that the cable comes out downwards. It is possible, but very difficult, to plug this in the wrong way round. Some drives come with their own power supply and some do not. Drives from Viglen with an integral power supply, can be identified by a suitable mains warning label on the back of the drive. Look at the plug on the end of the power lead to see if it is a mains plug or not.

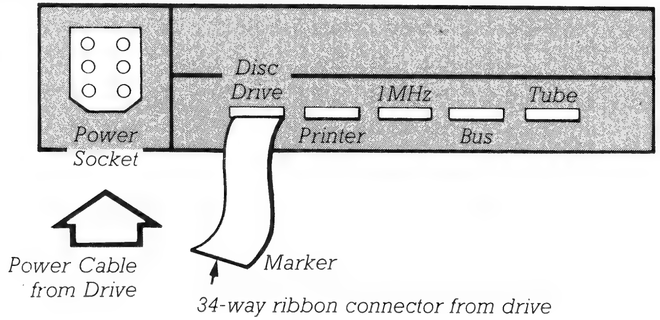
If the lead has a mains plug

This is the only kind of disc drive that should be plugged into the mains supply directly, as they contain transformers to produce the necessary low voltage.

If the lead does not have a mains plug

If the power lead to the drive does not have a mains plug on the end then it does not contain a transformer and runs from the low voltage supply in the computer itself. Never plug this type of drive into the mains as it is extremely dangerous and would damage the drive and probably the computer itself beyond repair. This type of power cable plugs into the socket underneath the computer labelled *'power out'*. This plug will only go in the right way round.

Some of the very first BBC computers came with linear power supplies which do not have enough capacity to



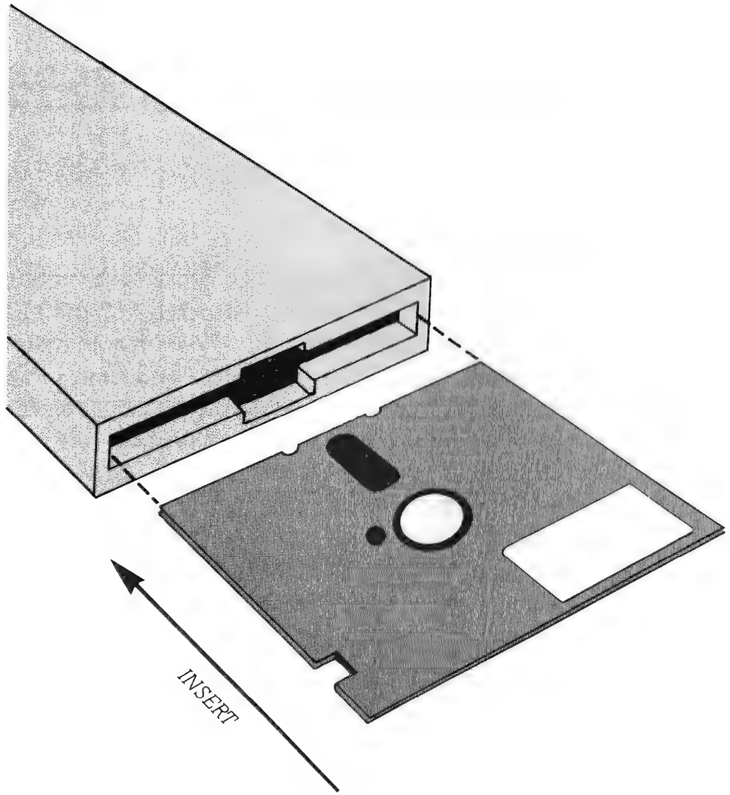
View from underside of the BBC micro (front side) power the drives as well as the computer. In fact they become quite hot just driving the computer. If you have one of these then you should have it changed first, or get drives powered from the mains. The old linear power supply can be identified by being made of a blackened metal, whereas the newer switched mode power supplies are a dull silver colour. Look at where the mains lead goes into the computer, as the power supply case is exposed here.

Switching on

If you're not familiar with using discs, read *Appendix B* on care of discs. In particular, don't switch on or off with a disc in the drive. Switch on, remove the protective cards which may be in the drives to prevent damage in transit and insert your utility disc.

5¼ inch discs must be inserted with the label uppermost (note that you cannot flip this disc over to use the other side) and the door of the drive must be shut. Depending on the make of drive, the door can be shut either by turning the flap on the front of the drive or by pushing the button on the front. 3 inch discs can be inserted with either the side A label or side B label uppermost. The utilities are on side A and thus to start off with you should insert the 3 inch disc with the side A label uppermost; the disc will automatically click into place.

Now hold the <shift> key down and press the <break> key, releasing the <break> key first. If all is well then the utility disc menu will be loaded in and you can proceed to format your spare discs.



Inserting a 5¼ inch floppy disc

5. Formatting and verifying discs

This section covers the use of the utilities disc to format and verify discs.

What does formatting mean?

At its simplest level formatting is something that you have to do to a new disc before you can use it. This is all that you actually have to know, but if you understand the following explanation then so much the better.

If you look on page 399 of the User Guide then you will see that on tape each block of data is surrounded by catalogue information, block identification, synchronisation bytes and error checking bytes. On the disc system the catalogue information for all the files is stored on the two sectors 000 and 001. Apart from this the sector on disc is very similar to the block on tape.

When you first buy a disc it is totally empty, like a blank cassette. The disc filing system needs the disc to be covered in empty sectors to find its way around. The hole in the disc was mentioned in *Chapter 2* as a means of the disc filing system knowing where the disc is. In fact, it needs more accurate timing than that. Each sector is preceeded by an identification field, which contains a number 0-9 to identify it, and a synchronisation signal. Even an empty sector must have an ID field or the filing system will not be able to find it in order to store data in it. Formatting consists of filling the discs with synchronised, identified sectors full of dummy data (&E5) and writing a completely empty catalogue into sectors 000 and 001.

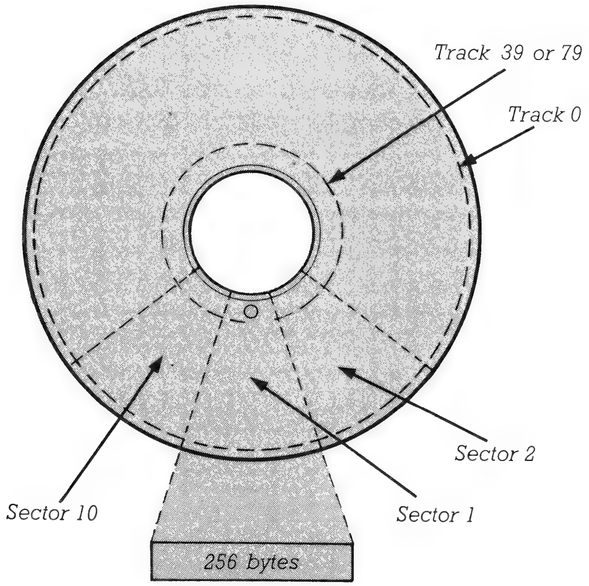
You can tell whether a disc has been formatted or not by putting the disc in drive 0 and typing '*CAT'. If the disc has been formatted then you will at least get an empty catalogue. If not, then the computer will say *'disc fault'* because it can't even find an empty catalogue on the disc.

How to format a disc

You have been provided with a utilities disc which contains a program to format discs.

1. Insert the utilities disc into drive 0 and press the <shift> key and the <break> key together, releasing the <break> key first. If you have a 40/80 track switchable drive then ensure that the switch is in the correct position. The utilities disc supplied is normally recorded on 40 track and thus the switch position should be set to 40.

2. You are given a menu of options, one of which is to format a 40 track disc and one of which is to format an 80 track



Format of the disc in terms of 'track' and 'sector'

disc. Decide which you require and press the right number. There is no need to press <return>.

3. You are asked which drive you wish to format. If you have a double drive then you can put your new disc into drive 1 and answer '1'. If you only have a single drive then you must answer '0'. To format the second sides of discs, use numbers '2' (one double sided drive) or '3' (twin double sided drives). Side 2 is the reverse side of side 0 and side 3 is the reverse side of side 1. You can only use side 2 or 3 if you have a double sided disc drive.
4. Now insert your new disc into the drive that you have specified. Be very careful to only insert new discs as the formatting process will erase any data that may be on the discs. If you have a 40/80 track switchable disc drive then you must ensure that the switch is now set to the position for the number of tracks that you want to format. When you have done this then press 'Y'.
5. The formatting will now proceed. You are kept informed of the progress by the number of the track being formatted being printed out in hexadecimal.
6. When this has finished then you will be asked if you want to format another disc. When you buy a batch of ten, it is a good idea to format them all at once to save messing about later on.
7. When you have finished then you will be returned to the menu. To use any of the other utilities on the menu you must of course re-insert the utilities disc. To get out of the menu you must press the <break> key.

Verifying a disc

Verification is no more than making sure that everything on a disc can be read from it and has not been corrupted. Using the verifier on a disc is a much more complete test than just reading all the files on a disc because it even checks the tracks that are not in use. Using the verifier is virtually the same as using the formatter. The procedure is as follows:–

1. Insert the utilities disc into drive 0 and press the <shift> key and the <break> key together, releasing the <break> key first.
2. You are given a menu of options, one of which is to verify a disc. Press the correct number, there is no need to press <return>.
3. You are asked which drive you wish to verify. If you have a double drive then you can put your disc into drive 1 and answer '1'. If you only have a single drive then you must answer '0'. To verify the second sides of discs, use numbers '2' (one double sided drive) or '3' (twin double sided drives).
4. Now insert your suspect disc. If you have a 40/80 track switchable drive ensure that the switch is set to the correct setting. When you have done this then press 'Y'.
5. The verification will now proceed. You are kept informed of the progress by the number of the track being verified being printed out in hexadecimal.
6. When this has finished then you will be asked if you want to verify another disc.
7. When you have finished then you will be returned to the menu. To get out of the menu you must press the <break> key.

The utilities disc contains other features which may be of interest to you. These are described in the menu.

6. Using discs to load and save programs

This short section will enable you to get started by explaining how to do the simple things that you have been used to doing with cassettes.

Loading and Saving

The LOAD and SAVE commands work in a similar manner to that which they do on cassette. There are four differences that you should know about:

1. The filename must not be longer than seven characters. Do not use any of the following characters in the filename:–

   ! * : # . <space>

   as these carry a special significance.
2. No messages are printed on the screen as the program is being loaded or saved. File storage and retrieval all happens so quickly that you don't need reassurance that it is working.
3. With the cassette system you could type 'LOAD""' or 'CHAIN""' to load the next program on the tape. This has no meaning on a disc as there is no particular order therefore you must give the filename when specifying a file.

Other commands

When you type '*CAT' the computer will print out a heading with information in it, which will be covered later, and a list of all the programs on the disc. The commands '*LOAD', '*SAVE', '*RUN', '*SPOOL' and '*EXEC' all work as before and are explained in *Chapter 41 of the User Guide* that came with the computer.

You may still want to use the tape filing system which you can do at any time by typing '*TAPE'. Having done this you can get back to the disc system by typing '*DISC'. You can do this within a program if you want to.

7. Drives, directories and file specifications

This section details how to tell the computer which drive you want to use in a disc operation, how to keep programs in separate groups called directories and how some commands can be made to operate on several files at once.

Drive addressing

So far, any saving and loading you have done has always been on drive 0. The computer assumes that you want to use drive 0 unless you tell it otherwise. If you have a second drive then it will be called drive 1. If you have double sided drives then the second side of drive 0 is drive 2 and the second side of drive 1 is drive 3. The second sides of the drives are given their own drive numbers despite being part of the same drive unit.

One way of telling the computer that you want to use drive 1 would be to change the 'current drive number' to 1. This is achieved by typing:-

```
*DRIVE 1
```

From then on the computer will use drive 1 whenever you LOAD or SAVE. This will continue until you press <break> or until you change it back by typing:-

```
*DRIVE 0
```

The full syntax for this command is:-

```
*DRIVE (<drv>)
```

Another way of accessing other drives is to include the number of the drive in the filename. If you type:-

```
SAVE ":1.PROG"
```

then the program will be saved on drive 1 with the name 'PROG' irrespective of which drive is the current one. If you take the disc out of drive 1 and put it in drive 0 then, to load the program in, you will have to type:-

```
LOAD ":0.PROG"
```

The colon and the fullstop serve to separate the drive number from the main filename. If you want to get a catalogue of what is on a disc in drive 1 then type:-

```
*CAT 1
```

The syntax definition of this command is:-

```
*CAT (<drv>)
```

If the drive number is omitted then the current drive is assumed. Note that you can now perform a '*CAT' from within a program. This was not possible with the tape system because the only exit from getting a catalogue was by means of the <escape> key.

In the header to the catalogue the computer tells you which is the current drive. It is important to realise that this is stored in the computer and not on the disc. Because it is given in the catalogue of the disc, one feels that it must be a parameter of the disc which is not the case. There are several things contained in the catalogue which are parameters held in the computer.

Directories

When the filing system stores the name of a program it also stores a single identifying character with the program name. The idea of this is that you should think of all the programs with the same identifying letter, say 'A', as being in a group or directory called 'A'. If a directory letter is not specified then the 'current directory' letter is used by default. When the <break> key is pressed then the current directory letter is set to '$'. The current directory letter can be changed by typing:-

```
*DIR (<letter>)
```

where <letter> can be any non reserved character. If you want to specify a program which is not in the current directory then you can type:-

```
LOAD "B.PROG"
```

which will load the program called 'PROG' in the B directory, even if the current directory is not 'B'. The filing system regards "A.PROG" and "B.PROG" as being different programs because they are in different directories. If you want to specify the drive number at the same time as the directory letter then the drive number comes first as in:

```
LOAD ":3.A.PROG"
```

which will load the file called 'PROG' from directory 'A' on drive 3.

An example of how a facility like this can be used follows. Imagine you've written a program which works out how much your car costs to run. You have two data files about the car which are called 'CAR1' and 'CAR2' and a file of costs called 'COSTS'. You then wish to use the same program on a friend's car. This creates a problem because if you call the files about his car by the same names then they will overwrite your files but if you call them something else the program will use yours instead.

The way round this is to use three directory letters, say 'Y' for your car, 'H' for his car and 'C' for the costs file. You will have five files in total; 'H.CAR1', 'H.CAR2', 'Y.CAR1', 'Y.CAR2' and 'C.COSTS'.

When, in your program, you want the data file about the car then just refer to it as 'CAR1' and 'CAR2'. Refer to the costs file in full as 'C.COSTS'. Before you run the program for your car, type:-

```
*DIR Y
```

or, to run it for your friend's car, type:-

```
*DIR H
```

When the program loads the car files, it will use the current directory that you have specified. If the current directory is set to 'Y' then it will read in the files about your car but if it is set to 'H' then it will read in the files about your friend's car. As the costs file is specified in full with the directory letter it will be loaded correctly even though it is not in the current directory.

The current directory letter is also found at the top of the catalogue and includes the drive number. You can combine the two commands:-

```
*DRIVE 1
*DIR Z
```

into:-

```
*DIR : 1.Z
```

which will have the same effect. The directory can be set to the default by just typing:-

```
*DIR
```

This will cause the directory to be '$' and the current drive to be 0.

Libraries

The command '*RUN MC', which runs a machine code program called 'MC', can be abbreviated to '*MC'. This is so that the disc can contain programs which are thought of as an extension to the operating system commands. When a program is run in this way then the filing system looks not in the current directory, but in the current library. This is also shown on the catalogue heading and can be changed with the command:-

```
*LIB (<letter>)
```

If the current directory is 'A' and the current library is 'B' then 'LOAD"PROG"' will load 'A.PROG' but '*MCPROG' will run 'B.MCPROG'.

File Specifications

Some commands can operate on many files at once. In order to specify which group of files that you want the command to operate on then you can use the single letter wildcard '#' or the multiple letter wildcard '*'. They are called wildcards because they can represent any letter that you like. A case in point is the command '*INFO' which gives information about a file or program. To find out about 'PROG' you type:-

```
*INFO "PROG"
```

and you will get a line of information which is described in *Chapter 8*. If you wanted to find out about all the files in the current directory beginning with 'P' and having four letters then you would type:-

```
*INFO "P###"
```

which says that the first letter must be 'P' but the other three can be anything. To find out about all the files starting with 'PR' and having any number of letters type:-

```
*INFO "PR*"
```

Here are some more example filespecs:–

*	*all the files in the current directory*
##	*as above but only those with two letters*
G.*	*all the files in directory G*
#.*	*all the files on the disc*

In the syntax definitions, if a command can only take a specific filename then it is shown as <fsp> but if it will operate on many files at once then it is shown as <afsp> which stands for 'ambiguous file specification'. The syntax for '*INFO' is:–

```
*INFO <afsp>
```

You can only use wildcard characters for the directory letter and the filename but not for the drive number.

8. Copying, renaming and deleting files

This section covers moving files around on the disc, making backups, locking files and finding the free space left on the disc.

Operating System Commands

One of the best features of the BBC Computer is how BASIC and the operating system have been kept separate with the intention that the operating system can be used with other languages. This means that the operating system should not retain any of the characteristics of BASIC. If BASIC is exchanged for PASCAL which is to be used with the same operating system then you would not expect to find little idiosyncracies of BASIC left behind. The commands prefixed with a ' * ' in BASIC which are passed to the operating system including all the disc commands can appear to exhibit strange behaviour to someone used to BASIC.

1. The operating system works in hexadecimal so the '&' prefix is not required when giving addresses in '*LOAD' and '*SAVE' operations.
2. BASIC uses the colon as a statement separator. However this is not used in operating system commands. The result of this is that you cannot have lines like these in your BASIC program:–

```
10  MODE7:*FX6,0:ON ERROR GOTO 600
20  IF A=5 THEN *FX10,20 ELSE *FX9,20
30  INPUT "IGNORE CHARACTER",A:*FX6,A
```

As soon as the ' * ' is encountered in lines 10 and 20 then the whole of the line is passed to the command line interpreter. BASIC then moves on to the next line. The 'ONERROR' and 'ELSE' statements would therefore be lost. In line 30 the variable 'A' has been included in an operating system command but the operating system knows nothing of BASIC variables.

3. BASIC differentiates between upper and lower case letters whereas the operating system does not. This means that you can type '*fx', '*tape', '*load', etc. A nasty twist to this is that, whilst the system puts upper and lower case characters in filenames in the catalogue, it does not regard them as different when deciding whether it has found a file or not. If you save a file as 'PROG' then you can

load it in as 'prog'. If you have a file called 'PROG' and you save another file called 'PROG' then the second would overwrite the first as the system will assume that it is just an updated version of the same file. However, if you save a file called 'prog' then it will STILL overwrite the first file even though you probably intended it to be a different filename.

4. It is not necessary to enclose filenames in quote marks as quote marks are a feature of BASIC. If quote marks are used then they will be ignored. The command:-

```
*INFO "#.PROG"
```

may equally well be written:-

```
*INFO #.PROG
```

Quote marks must be used if the string contains spaces.

Copying and renaming files

Before we start there are three commands which use the same memory space as BASIC programs. These are '*COPY', '*BACKUP' and '*COMPACT'. If you want to use any of these facilities then make very sure that you don't want whatever is stored in the computer's memory as it will be lost.

To copy a file called 'PROG' from one disc to another put the source disc in drive 0, the destination disc in drive 1 and type:-

```
*COPY 0 1 PROG
```

The original will be left intact on drive 0. You can copy several files at once by using the ambiguous file specification. To copy all files in directory 'A' to the second disc for example, type:-

```
*COPY 0 1 A.*
```

The full definition of this command is:-

```
*COPY <src drv> <dest drv> <afsp>
```

The file is copied from drive <src drv> to drive <dest drv>. If you have only one drive then you can specify this drive for both. The command will ask you to insert each disc in turn and press a key before continuing.

If you want to have a second copy of a file on one disc so that you can edit one and keep the other as a backup then you must do this in three stages. Firstly, copy the file onto a spare disc as described above. Secondly, RENAME the file on the main disc with the following command:-

```
*RENAME PROG OLDPROG
```

which will change the name of the file from 'PROG' to 'OLDPROG'. The syntax of this command is:-

```
*RENAME <fsp> <fsp>
```

Thirdly, copy back the file 'PROG' from the spare disc which will not overwrite the file 'OLDPROG' as it now has a different name. To make a backup of the whole disc use the command:-

```
*BACKUP 0 1
```

The full definition of this command is:-

```
*BACKUP <src drv> <dest drv>
```

As this command completely destroys anything that was on the destination disc you must have previously 'enabled' the command by typing:-

```
*ENABLE
```

as the immediately preceding command. As with the '*COPY' command you can '*BACKUP' with only one drive by saying:-

```
*BACKUP 0 0
```

If you have a 40/80 track switchable drive you can copy from one track format to the other using the *COPY command. You must of course ensure that the switches are set up for the correct number of tracks. For a single drive the switch position must be changed each time the destination and source discs are swopped over. You cannot use the *BACKUP command to copy from one track format to another though you can use the *COPY command with the <afsp> set to *.* (for example *COPY 0 1 *.*) to achieve the same result. Note that the auto-start option is not copied.

Locking files

There are two ways of protecting a file from being overwritten. The whole disc can be protected by putting a label over the write protect notch,or, for 3 inch drives, sliding the red tab to one side. When you buy spare 5¼ inch discs you will get a batch of black or silver labels for this purpose. Don't use clear tape because some drives detect the label optically.

Alternatively a file can be LOCKED in software by labelling the file as such. If you want to protect a file called 'PROG' type:-

```
*ACCESS PROG L
```

When you catalogue the disc an 'L' will have appeared by the filename. If you try to save another program called 'PROG' which would overwrite it then the system will not let you. You will also be unable to delete the file until you unlock it. To unlock the file, type:-

```
*ACCESS PROG
```

The syntax of the command is:-

```
*ACCESS <afsp> (L)
```

This shows that you can lock or unlock a group of files with one command. If the 'L' is present then the files are locked but if absent then they are unlocked.

Deleting files

There are three ways of deleting a file not counting over-writing it. They are:-

```
*DELETE <fsp>
*DESTROY <afsp>
*WIPE <afsp>
```

The '*DELETE' command is for deleting one file only. The

'*DESTROY' command will delete a group of files but, because this command is so drastic, it requires the '*ENABLE' command to be issued first. The command will list the group of files and ask you if you still want to delete them all. The '*WIPE' command will also delete a group of files but will ask you about each one in turn before deleting it.

Disc organisation

If you delete a file then a free space will be left behind. After a period of juggling files about, there may be lots of small free spaces scattered around the disc. If you want to save a file that is bigger than any of these the system will just say *'disc full'* because it will only save the file if there is a space large enough to hold it intact. The operating system is not intelligent enough to save one part in one place and the rest in another. If this happens then you must save the file somewhere else first and then compact the disc by typing:–

```
*COMPACT <drv>
```

It is vital to save your file somewhere else first because this command uses all of the memory as a temporary store. This command eliminates the free spaces and moves all the files on the disc up to one another which creates one large free space at the end. You can then copy the file from the other disc into this free space. This command also tells you in hexadecimal how many free sectors there are on a disc. A file always uses a whole number of sectors. Sectors 000 and 001 are used for the catalogue therefore the first file will start in sector 002. If the file uses two and a half sectors it will be found in sectors 002, 003 and half of 004. The other half of 004 is not used and the next file will start in 005. Therefore the free space is always a whole number of sectors.

As the compacting command proceeds, it gives the information line for each file just like the '*INFO' command.

For example:–

```
$PROG     L FF1900 FF801F 0000FB 017
```

The information is as follows:–

$.PROG	*directory and filename*
L	*indicates that file is locked*
FF1900	*load address of file*
FF801F	*execution address of file*
0000FB	*length of the file in bytes*
017	*sector in which file starts in hex*

followed by the free space message. Disc organisation is covered more fully in Chapter 11.

9. Other useful features

This section details the remaining commands available to the disc user. Some of these utilities will also work on cassette files despite being part of the disc filing system.

Help commands

To get a list of the ROMs plugged into your computer, type '*HELP' which will cause any ROMs which have this title and version numbers. Under the heading 'DFS' you will see 'DFS' and 'UTILS'. This means that you can type either of the following:-

```
*HELP DFS
*HELP UTILS
```

to get lists of the syntax definitions of the filing system commands and the utility commands.

Disc title and version number

It is possible to give a disc a title of up to 12 characters. The only use of this facility is that it appears at the top of the catalogue listing. To give a disc a title, type:-

```
*TITLE <title>
```

This will cause the title to be written to the disc. The title can only contain spaces if the string in enclosed in quotes.

When the disc is formatted, the two digit number displayed next to the title in the catalogue is set to zero, (appearing as '00'). This number is increased by one each time the disc is accessed for a write operation. When it gets to '99' the next write operation takes it around to '00' again. If you keep backups of discs then this may help you keep track of which disc is which.

Hexadecimal listing

The utility command '*DUMP <fsp>' will produce a hexadecimal dump of the file on the screen with the corresponding characters, if they are printable, on the right. This is useful either if you are unsure about the structure of a file or when debugging a file handling program.

Auto-booting

When you press the <shift> key and the <break> key together, releasing the <break> key first, then the computer will read an option from the disc and act accordingly. Here is a table of commands to set these options and their effects on the special file called '!BOOT':-

Option	**Action**
*OPT4,0	*no action*
*OPT4,1	**LOAD !BOOT*
*OPT4,2	**RUN !BOOT*
*OPT4,3	**EXEC !BOOT*

The '*OPT4,n' command must be issued with the disc in the drive because the option number is written onto the disc. The current value of this option is shown at the top of the catalogue heading.

Possibly the most useful of these options is '*OPT4,3' because this enables you to set a BASIC program to run automatically. You can build up the '!BOOT' file to contain the instruction 'CHAIN"PROG"' by typing:–

```
*BUILD !BOOT
```

When you are given line number 1 type the line 'CHAIN"PROG"'. You will then be given the line number 2. As you have finished, press the <escape> key and the file will be written to the disc. This is similar to use AUTO line numbers in BASIC. If you want to check the contents of the '!BOOT' file or any other file that has been built up with '*BUILD' then you can either use:–

```
*LIST <fsp>
```

which will list the file and give you the line numbers or, to list the file without the line numbers, use:–

```
*TYPE <fsp>
```

Types of text file

It is most important to realise that there are three different ways of storing text in a file which are not compatible with each other. The best way to describe these three groups is to list the commands which use them:–

Group 1:	*BUILD *EXEC *LIST *TYPE *SPOOL
Group 2:	LOAD SAVE (for BASIC programs)
Group 3:	PRINT# INPUT# (in BASIC programs)

It may seem strange not to use the same format for all cases but there are good reasons for not doing so. The group 1 commands store text in the simplest way which is compatible with some word processors (e.g. WORDWISE). For the sake of compactness BASIC programs are stored with tokens for keywords. PRINT# and INPUT# have to store the strings in such a way that they can be differentiated from numbers as these can be stored together in the same file.

10. Filehandling in basic

The user guide that came with the computer covers the use of the BASIC file handling commands. However, this chapter may help to make things a little clearer.

Structure of Files

The DFS stores a file as a long string of bytes and knows nothing of the structure of its contents. It is the BASIC interpreter which gives the file its structure. The BASIC programmer can store three different kinds of data in a file. These *'types'* are strings, integers and real numbers.

The following passage describes in detail the creation of a file with one data item of each type in it.

Printing to a file

To first create the file the BASIC command 'X=OPENOUT ("FILE")' is used. The BASIC interpreter asks the DFS to set up a new file called 'FILE'. The DFS checks the disc catalogue to make sure that it is okay and then allocates a channel number which is returned to the BASIC interpreter. *Note that if a file of that name already exists it will be deleted.* Up to five files can be open at once, where the channel number identifies which file is to be used. In the above command the BASIC Interpreter puts this channel number into the variable X which is used to identify the file in all future references to it. To refer to the file by its filename each time would be slower and also use more memory space. To put a string into the file, the following statement is used:-

```
PRINT#X,"VIGLEN"
```

As mentioned before, the DFS only deals in bytes and each byte will only hold one character. Because of this, the BASIC interpreter sends a sequence of bytes to store the whole string. The first byte sent is &00, which identifies the following bytes as being of type *'string'*. The next byte is the number of characters in the string, which in our case is &06. The next six bytes are the string itself, one byte for each character and in reverse order.

To store an integer, the statement used is:-

```
PRINT#X,A%
```

The BASIC interpreter first sends the byte value &40, which identifies the following bytes as being of type 'integer'. The number itself is stored in four bytes with the most significant byte first.

To store a floating point number, the statement used is:-

```
'PRINT#X,A'
```

The BASIC interpreter fist sends the byte value &FF, which identifies the following bytes as being of type *'floating point number'*. The number itself is stored in normal floating point format in five bytes.

Having written the data items to the file it must be closed by using the statement:-

```
CLOSE#X
```

The need to open and close files may seem a nuisance at first, but there is a good reason for it. The DFS keeps a quarter of a 'K' (256 bytes) of an open file in a buffer. A buffer is just a name for a section of memory set aside for some purpose. In the example just presented, it would be very slow if the DFS were to store each byte on the disc one at a time.

Instead, it saves them up in the buffer and only stores them on the disc when the buffer is full. This makes the need to close the file more obvious. At any one time there is likely to be part of the file still in memory which still requires writing onto the disc to complete the operation. Correspondingly, when a file is opened for input then 256 bytes are read from the disc and stored ready for the next few read operations.

The structure of the file can be examined by using the utility command '*DUMP' as follows:-

```
>*DUMP FILE
0000  00  06  4E  45  4C  47  49  56   ..NELGIV
0008  40  00  00  00  05  FF  00  00   @.......
0010  00  00  80  **  **  **  **  **   ........
>
```

This demonstrates the internal structure of the file.

Inputting, pointer and extent

The data items are read back in a similar way. This section explains the values of two variables 'PTR#' and 'EXT#'. To open the file for input, type:-

```
Y=OPENIN("FILE")
```

This reads the first sector from the disc. In our example it will load the whole file as it is less than one sector long. There are now two functions which can be used, which can also be used on an output file. The function 'EXT' which stands for extent, must be given the channel number, as for 'PRINT' and 'INPUT'.

```
PRINT EXT#Y
```

This gives the length of the file in bytes not the number of items of data which, for a file in input, will remain the same. In this case it will be 19. The DFS keeps track of where it is in the file by means of a pointer, to which one has access by means of the function 'PTR' which is used in the same way as 'EXT'. If the following statement is executed then 0 will be printed because no bytes have been read from the file yet:-

```
PRINT PTR#Y
```

The following statement sets A$ to the string 'VIGLEN', with the value of 'PTR#Y' changing to 8 as 8 bytes will be read from the file:-

```
INPUT#Y,A$
```

The following statement sets A to 5 and moves 'PTR#Y' on to 13:-

```
INPUT#Y,A
```

Although A is a floating point variable and it has read in an integer, this does not upset the BASIC interpreter. When BASIC requested the next byte from the DFS, it was &40 which identified the following data as being of type 'integer'. BASIC then received the next four bytes from the file via the DFS, converted them into a floating point number and stored this number in A. Similarly, if the next number, which is floating point, is read into an integer variable then it will first be converted to an integer by rounding down. The following statement will therefore put 0 into A%:-

```
INPUT#Y,A%
```

As we have now reached the end of the file 'PTR#Y' and 'EXT#Y' will now be set to the same value and 'EOF#Y' will now be set to 'TRUE'(-1).

We must finish the operation by closing our file by typing:-

```
CLOSE#Y
```

This flexibility in the type of number is the only kind which is allowed. If, for example, you try to read a number when the next item is of type *'string'* then a *'type mismatch'* error will be generated. This will also occur if the identifying byte is not &00, &40 and &FF.

It is possible to set the value of 'PTR#Y' to a number which is less than 'EXT#Y' thus giving you a truly random access file. This will cause the DFS, if necessary, to read a different sector from the disc. Be very careful with this facility as strings, floating point numbers and integers are stored in differing numbers of bytes, making it difficult to calculate the position of any particular item.

Putting and getting bytes

It is possible to deal with bytes directly. A byte is not a data type as such, and has no identifier. A byte can be read from an open file by using:-

```
B=BGET#Y
```

which puts the value of the next byte into B. A byte can be put into an open file by using:-

```
BPUT#Y,B
```

not forgetting that B must not be greater than 255 or the number will not be stored correctly.

In the later versions of the BASIC interpreter (i.e. BASIC2) the command OPENUP can be used to open a file for both reading and writing (particularly important for random access). For example X=OPENUP("FILE"). Unlike OPENOUT this will not delete the file "FILE" if it exists.

Example Program

Here is a program which may be of use to you and also demonstrates many of the features described above:-

```
 10  INPUT"NAME OF FILE TO BE PRINTED?"A$
 20  IN=OPENIN(A$)
 30  REPEAT
 40  B=BGET#IN
 50  PTR#IN =PTR#IN-1
 60  IF B=0 THEN INPUT#IN,A$:PRINT"STRING:
     "A$:GOTO100
 70  IF B=&40 THEN INPUT#IN,A%:PRINT"INTEGER:
      "A%:GOTO100
 80  IF B=&FF THEN INPUT#IN,A  :PRINT"FLOATING:
     "A  :GOTO100
 90  PRINT"NO SUCH TYPE":CLOSE#IN:END
100  UNTIL EOF#IN
110  CLOSE#IN
120  END
```

Description

This program prints out the contents of a file created from BASIC, without needing to know what kind of data is in the file. It looks to see what type of data is present and reads it in correctly.

Line 10 requests the name of the file and line 20 opens it for input. Lines 30 and 100 cause the middle part of the program to be repeated until the end of the file is reached: Line 40 reads the first byte from the file, which will be the type identifier for the first item. This uses up the byte which will be needed by BASIC to read the item in, therefore the pointer must be put back by 1. This is achieved by line 50. Lines 60, 70 and 80 check the byte to see what type the item is, read it into the correct type of variable and print it out with the type identified. If the byte does not represent any known type then the program goes straight through to line 90 which closes the file with an error message. After reading each item the program goes back to line 40 to identify the next item. If the EOF flag is set, indicating that there are no data items left, then the file is closed and the program stops.

11. File management

The DFS does a lot of the disc organisation for you without you being aware of what it is doing. It is not perfect however, and an appreciation of how it works will help you to get the most from your system.

Storing areas of memory

The DFS will only store a file in a continuous block. More sophisticated filing systems can split files up with links to join them together. Although this eliminates the need for such commands as *COMPACT, it can actually make files slower to load as their parts may be scattered all over the disc. This means that, after a period of use, your disc may be left with lots of gaps on it without you realising it.

When a block of memory is saved, whether it is data or a program, it looks for the first space which is big enough to take the file intact. For example, say you save program 'A' and then save program 'B'. You then load 'A' back in again to add a few lines, then save it back to the disc. In this case the new program 'A' may easily take up one more sector than it did the first time. The DFS deletes the original 'A' and saves the new 'A' AFTER program 'B'. This leaves a gap before program 'B' of the size that 'A' used to be. If a shorter program 'C' is then saved, this will be stored in this spare gap but, if it is shorter than 'A' was, then there will be a small gap between 'C' and 'B'.

In this way it is easy for a disc to get 'messy' so it is wise to '*COMPACT' it regularly. If the disc in the example above were to be compacted then 'C' would be left where it is, 'B' would be moved up next to it and 'A' would be moved up next to 'B'.

Files Opened for Output

If a file is opened for output then the DFS has no idea how much data you are going to put in it, making it difficult for it to know where to put it on the disc. If an output file is in a gap between two other files there is a limit to how much data can be put in. The output file is still constrained by the need to use a consecutive block of sectors so, if more data is put into the file than there is room for between the two files, the *'can't extend'* error message is generated. It would be silly to start writing an output file into the first gap it finds because, if say it was only one sector long, the chances of failing to extend would be very high. On the other hand, it can't reserve an enormous amount of disc space because if you tried to open several files for output at once the disc would soon become fully allocated and a 'disc full' error would be generated.

The compromise figure chosen by the writers of the DFS is &4000 bytes. This is just under a sixth of the whole disc space on a 40 track drive. It is still possible to get the above errors generated but careful planning will help you to avoid them.

Avoiding 'Can't extend'

This is caused when an output file is being enlarged and the DFS comes to a sector on the disc which is in use. If there is nothing on the disc after the file in question then the DFS will continue extending the file, even after the &4000 bytes first allocated, until it gets to the end of the disc. If you open two or more files for output the second will be placed &40 sectors on from the first. Putting more than &4000 bytes into the first will therefore cause it to run into the second, generating this error.

Although &4000 bytes of disc space are allocated, when the file is closed the actual amount used is recorded in the catalogue and any unused space is then freed. If a file which already exists is opened for output then the DFS will use the same space as the old file thus overwriting it. If the new file is longer than the old one, such that it needs another sector but the space after the old file is in use, then this will also cause this error.

Both of these problems can be avoided by allocating the amount of space you need in advance.This can be done by using the '*SAVE' command to save a dummy file of the right length. If you think you need &6000 bytes in file'A' and &200 bytes in file 'B' then save dummy files as follows:-

```
*SAVE FILEA 0 6000
*SAVE FILEB 0   200
```

This will create dummy files of the correct length. When the files are opened for output then the whole of this space and no more will be allocated instead of the arbitary &4000, thus avoiding the error.

Avoiding 'Disc full'

This error can be avoided in much the same way. If you only have a little space left on the disc and you want to open five files for output then the DFS will try to allocate &4000 bytes for each. If there is room to save five dummy files in advance with the right names and the right lengths then they can all be opened at once without generating an error.

Converting programs from tape to disc

This section provides a general guide for copying your own programs that you have previously saved on cassette to a disc. Note that it is illegal to copy a program that is protected by a copyright, though some software producers allow one backup to be made for an individual's own use. Some software houses offer a disc upgrade service whereby the original program on cassette is exchange with an original disc version for a small fee.

General procedure

It would be very nice if a universal procedure could be described which would enable any program to be transferred to disc in a form in which it will run. Unfortunately all programs are different and so no universal procedure is possible. This section should help you to convert as many as possible. The first step is to get the tape files onto the disc. Most files can be transferred to disc using the following general procedure but there is guarantee that they will run.

The simplest program to convert from tape to disc is the small basic program (i.e. one that would fit in the memory area from &1900 to HIMEM).

This is performed as follows:-

1. Type '*TAPE'
2. Type 'LOAD "<name of file on cassette>"
3. Type '*DISC' and insert a formatted disc in the drive.
4. Type 'SAVE "<name of file on disc>"

For all other programs the method outlined below can be used. Please note that all addresses are in HEXADECIMAL. They may not be prefixed with an '&' because load and save commands assume hexadecimal notation.

1. Type '*TAPE' followed by '*OPT1,2'. This enables the tape operating system and causes full file information to be printed when a file is loaded.
2. Type '*LOAD"" 1900' and play the tape. This causes the file to be loaded at address 1900. When the last block is being loaded a line of information will be printed. Make a note of this for later. The information is in the following order:

   Filename
   Number of blocks
   Length of file
   Start address
   Execution address

Both addresses are 8 hexadecimal digits long. The first four are only important if you have the TUBE and a second processor.

3. Type '*DISC'. This reactivates the disc filing system. Put a disc which has been formatted into drive 0.
4. Save the file to disc by typing '*SAVE' followed by:-

   Filename
   1900 (the address where you loaded the file)
   +LLLL (where LLLL is the length of the file)
   EEEE (where EEEE is the execution address of the file)
   RRRR (where RRRR is the start address of the file)

   Here is an example:-

   ```
   *SAVE "PROG" 1900 +2A80 801F 0E00
   ```

5. If the program is a suite of files then copy the other parts in the same way.

The above procedure loads the file at memory location 1900 and then saves it to disc. When saving to disc a relocation address is specified which is the real start of the file, rather than 1900 where it was stored temporarily.

Snags

If you are lucky and all the parts of the program are short and in BASIC and have short filenames then CHAIN'ing the filename of the first file will successfully run the program. Although the start address of the program is supposed to be E00, a BASIC program is relocatable so that if it is loaded at 1900 it will run.

There are several things that may go wrong. The first program may run the second by saying 'CHAIN""' meaning the next program on the tape. The DFS will not know which the next program is and so you will have to change the instruction to include the filename of the next file.

The tape files may have the same name or the same name in lower case letters. In either case trying to save the second file will overwrite the first. If this happens the first file should be saved under a different name. If you save a subsequent file under a different name, perhaps because its filename is too long, you will have to change the reference to the file in the program that loads it.

Often the last file in a program suite is machine code. You can tell a machine code file by the fact that its execution address lies within the program itself and by the fact that it is '*LOAD'ed and 'CALL'ed or '*RUN'. A BASIC program has 801F or 802F as its execution address. Unlike BASIC, machine code is NOT relocatable and so if it is loaded at the wrong address it will not run.

Relocation

There are several reasons why you may need to relocate part of the program suite, usually connected with having to keep the area of memory between E00 and 1900 clear for the disc system.

Here are some possible reasons:-

1. A BASIC program is too long. You get the *'Bad MODE'*

message when you run the program.

2. A BASIC program needs room for variables strings and arrays. You get the *'No room'* message while the program is running (usually just as you've found something new in an adventure!).
3. The program is in machine code located at E00 and so won't run at 1900.

The relocation of the last program must be done in the previous program which will hopefully be written in BASIC. If this last program is in machine code, list the previous program and work out where the last program is intended to be. If the '*LOAD' command has an address specified then this will override the start address stored with the file. We used this feature earlier to load files at 1900 regardless of the start address. If this clashes with the disc space then alter the program to load the last file at 1900, or possibly 1100 as described later on. Change the filing system with the '*TAPE' command and move the file to where it is supposed to be. When the file is in the right place it can be run with the same 'CALL' command. For BASIC programs which are too long change the 'CHAIN' command for a '*LOAD "filename" 1900' command, and then relocate it. It must be run using the technique described later on in this section.

The indirection operator '!' is very useful in relocation as you can move four bytes at a time. Here is an example of how to move a file which is 2345 bytes long from 1900 where you had to load it to E00 where it can be run:-

```
50  *LOAD"FILE" 1900
60  *TAPE
70  FOR I%=0 TO &2345 STEP 4:I%!&E00=I%!&1900:
    NEXT
```

Extra files

You must keep track of what areas of memory are in use at any time. If a program has been 'CHAIN'ed without changing PAGE then it will be located at 1900 so if this program loads a file in at 1900 the original program will be lost and it will crash. Suppose you have just one big file which will not run from 1900. In this case you must create two extra programs. The first program simply changes the value of PAGE to something large like &7800 and 'CHAIN's the second program. You must do this in MODE 7 which allows you memory up to 7000. The second program will then automatically load at this high location. The second program can then '*LOAD' the real program at 1900 and relocate it without being overwritten.

The remaining parts of this section will provide further information to help you cope with the changes from a cassette to a disc system.

Change in 'PAGE'

The 'PAGE' value having changed from &E00 to &1900 in the disc system means that in the highest resolution modes you are left with only 5.75K of RAM for your BASIC programs, strings and variables.

This is because the DFS requires room for a buffer for each file that you have open for either input or output. One sector of each file is held in RAM for quick access which reduces the amount of free memory for programs. The DFS also adds room to hold the catalogue information for the disc in use.

This may mean that you have to reconsider how you write your programs. You have exchanged a small amount of very fast access memory for a large amount of medium access disc 'memory'. If you have a program which has a lot of 'DATA' statements at the end then it may be better to hold the data on a separate disc file which is to be read in by the main program.

If the program can be split up into separate functions then it may be better to write it as several separate programs which can 'CHAIN' each other. If one program chains another one then all the dynamic variables will be lost with only @% and A% to Z% keeping their values. If the programs need to communicate values then these values must be saved in a file by the departing program in order to enable them to be loaded in again by the arriving program. The variables 'TOP', 'LOMEM', 'HIMEM' and 'PAGE' can be used to find out how much memory has been used. Starting from the bottom, 'PAGE' is the address of the start of the BASIC program. 'TOP' is the first free memory location after the end of the BASIC program. 'LOMEM' will be the same as 'TOP' unless you change it yourself and this is where BASIC stores its variables. 'TOP' is always at least 2 greater than 'PAGE' because even an empty program uses 2 bytes for the 'end of program' marker. 'HIMEM' is the address of the highest free memory location and varies depending on which screen MODE you are in. The screen uses memory from the very top, that is &7FFF, down to 'HIMEM'. The more memory the screen MODE uses then the lower 'HIMEM' will be. BASIC uses this space downwards to store information about procedures during execution. This is why you cannot change MODE inside a PROCEDURE or FUNCTION.

If you get the 'No room' message, it means that 'LOMEM' and 'HIMEM' are so close together that there isn't room for all the variables defined in your program. You must either shorten the program, use less variables or use shorter variable names.

Loading at &1100

One way of claiming back some of the memory lost to the DFS is to change 'PAGE' back to &1100 before loading your program. This will give you 2K of your memory back but does have disadvantages.

It works because the DFS only uses the RAM it has claimed if the disc functions are used fully. If you just want to load the program then the DFS only needs 3 pages (&300 bytes). It uses two for the catalogue in pages &E00 and &F00, as well as one for the file that is open to load or save the program itself, which is at &1000. If you were to open another file, either within the program or by trying to load the program as part of a !BOOT file then this will corrupt the program because another page of memory will be needed.

The second disadvantage is that if you press the <break> key then the program will be corrupted. The DFS sets

'PAGE' to &1900. BASIC then puts an 'end of program' marker at that address, which will be in the middle of your program and may make it unlistable.

The third disadvantage is that this feature is totally undocumented. Initially this may not appear to matter but it does mean that you cannot guarantee that it will work on all machines.

Principles of Loading and Shifting

If your program doesn't need the DFS after it has been loaded then you can disable the DFS and move your program into the memory it was using. This is probably the best way of loading games programs that only require loading from the disc and have nothing to do with file handling. Here is an example to show you how this would work with a large program.

The first thing you must do is to separate out anything that is not needed for the actual game and put it in an introduction program. You may have noticed that most commercial programs are done this way. These are the sort of things that can go into an introduction:-

(i) *title pages*
(ii) *instructions (these take up a lot of space)*
(iii) *character definitions*
(iv) *envelope definitions*
(v) *setting up game parameters*
(vi) *function key definitions*

Although BASIC loses its variables when chaining from one program to another, some other types do not get lost. Character definitions are not erased therefore it is quite safe to have all those 'VDU23,224,255. . . etc.' commands that use so much memory situated in the introduction program. Envelope definitions are not lost therefore all the 14-parameter ENVELOPE commands can also be put in the introduction. The single capital letter integer variables are not cleared so these can be used to set up game parameters for use by the second program. You could use N% for the number of players and S% for the speed of the bombs. The wordy questions that set these up can be safely put into the first program where space doesn't matter so much. As function key definitions are preserved these may also be in the introduction.

We now have a program in two parts. The first program does all the introduction and setting up with its last instruction being to 'CHAIN' the main program. The main program may now be short enough that it will fit into the remaining space. If it is still too big then some memory shifting may be necessary. This makes use of the fact that the space used by the graphics modes is not needed while the program is being loaded and the space used by the DFS is not needed while the program is running. The big program can be loaded into the space that is always free for programs (&1900 to &2FFF) and can overflow into the space used by the graphics modes (&3000 up). The DFS is disabled by the '*TAPE' command so that the program can be moved down to the space that was free with the tape operating system

(&E00 up). The program is now free to use the graphics modes as the top of the program has been moved down below &2FFF.

An Example Suite of Programs

Here are a set of three programs which operate in this way. These are so short that they do not need this technique but they adequately demonstrate the principle.

Program 1: Filename PROG1

```
 10  PRINT"This is the introduction program"
 20  VDU23,224,&FF,&7F,&3F,&1F,7,3,1
 30  ENVELOPE 1,1,4,-4,4,10,20,10,127,0,0,-5,126,126
 40  INPUT "HOW MANY PLAYERS",N%
 50  *KEY0"RUN M"
 60  MODE7
 70  PAGE=&7000
 80  CHAIN"PROG2"
```

Program 2: Filename PROG2

```
 10  *LOAD PROG3 1900
 20  *TAPE
 30  FOR I%=0 to &5700 STEP 4
 40  I%!&E00=I%!&1900: NEXT I%
 50  PAGE=&E00
 60  *FX138,082
 70  *FX138,0,85
 80  *FX138,0,78
 90  *FX138,0,13
100  END
```

Program 3: Filename PROG3

```
 10  @%=0
 20  MODE2
 30  PRINT"PAGE IS NOW &"~PAGE
 40  PRINT"There are "N%" players."
 50  SOUND1,1,100,60
 60  FOR I=1 TO 500
 70  COLOUR RND(7)
 80  PRINT CHR$(224);
 90  NEXT I
100  COLOUR 7
110  END
```

(i) The first program runs normally, doing all the introductory things described above.

(ii) In line 70 of the first program the value of 'PAGE' is set to a very high value so that there is plenty of room underneath where &7000 is a suitable value. The first program must only use MODE7. 'PAGE' is set this high so that an intermediate program can be loaded here to control the loading and shifting of the main program. Line 80 of the first program can then 'CHAIN' the intermediate program.

(iii) Line 10 of the second program loads in the main program starting at location &1900. There is plenty of room for it because the graphics modes have not been selected yet.

(iv) Line 20 of the second program then disables the DFS by using the '*TAPE' command. The memory from &E00 upwards is now free.

(v) Lines 30 and 40 now shift the main program down using the '!' operator which moves 4 bytes at a time from &1900 upwards where it was loaded, to &E00 upwards where it will run. Although &5700 bytes have been moved in this example, it may be unnecessary to move this amount of memory. Find the length of the actual program in bytes by using the '*INFO' command and use this value instead. This will reduce the time taken to shift the memory down. If in doubt, leave the &5700 as it is as it will certainly work, although it may take a few seconds.

(vi) In line 50 'PAGE' is set to &E00 in preparation for running the main program.

(vii) In lines 60 to 90 the characters 'R', 'U', 'N' and <cr> are inserted into the keyboard buffer to save the user from having to type 'RUN'.

(viii) The intermediate program ends and the BASIC interpreter finds 'RUN' in the keyboard input buffer so that it can run the main program. As the program is running from PAGE &E00, so it can use the higher graphics modes.

The variable 'TOP'

It may seem strangely contrived to put the 'RUN' command at the end of the second program into the keyboard buffer instead of just putting 'RUN' into the program. This is because the end of any program is defined by the value of 'TOP'. 'TOP' is used by BASIC to know where to store variables and strings and to issue the *'Bad MODE'* message if there isn't enough room. 'TOP' is only set properly under the following circumstances:–

(i) *OLD*
(ii) *LIST*
(iii) *END*
(iv) *LOAD (NOT *LOAD)*
(v) *CHAIN*
(vi) *at the end of a program*
(vii) *when editing or inserting a line*

You may recognise these as the times when the *'Bad Program'* message occurs. The computer searches from 'PAGE' onwards until it finds the end of the program in order to set 'TOP'. If it can't make sense of the program then it says *'Bad Program'.* When the second program is running, 'TOP' is set to the end of it because the 'CHAIN' command was used to run it. The problem is that the third program is loaded using '*LOAD' where neither setting 'PAGE' to &E00 nor saying 'RUN' will set 'TOP' to the end of the main program. The main program will start to run but, because 'TOP' is still set to the end of the second program, BASIC thinks there is no room to change MODE and the *'bad MODE'* or *'No room'* error will be generated.

To reset 'TOP' to the true end of the main program we must do one of the things in the list in *Chapter 12*, the easiest way being to allow the second program to actually end. When this happens BASIC starts searching from 'PAGE' to find the end of the program. As 'PAGE' is set at the beginning of the main program, it finds the end of this main program. Only then does it find the 'RUN' command that was previously stored in the keyboard buffer.

Machine code programs

This refers to programs that must be executed using the '*RUN' command. If you have written a program which is entirely in machine code then the load and shift method will still work but, instead of inserting 'RUN' into the keyboard buffer, you should use 'CALL' on the execution address. Find the start address, length and execution address of the program by using the '*INFO' command, ensuring that the second program moves it to exactly the right address. Machine code programs will not work if they are not in the right place. Then, instead of allowing the program to end and putting 'RUN' into the keyboard buffer, use 'CALL&nnnn' where 'nnnn' is the start address of the program.

13. Summary and syntax of DFS commands

This section lists all the disc filing system commands in alphabetical order and gives the syntax, the minimum abbreviation and a brief description of each.

- *ACCESS <afsp> (L)
  Minimum abbreviation: *A.

Locks (L) or unlocks a file.

- *BACKUP <src drv> <dest drv>
  Minimum abbreviation : *BAC.

Copies all the data and files from one disc to another. This command must be preceded by a *ENABLE command.

- *BUILD <fsp>
  Minimum abbreviation : *BU.

Creates a file directly from keyboard by causing all information entered from the keyboard to be stored in the file <fsp>.

- *CAT (drv)
  Minimum abbreviation : *.

Displays the catalogue from the disc.

- *COMPACT (drv)
  Minimum abbreviation : *COM.

Collects all the files on a disc into a continuous sequence leaving all the free space at the end.

- *COPY <src drv> <dest drv> <afsp>
  Minimum abbreviation : *COP.

Copies all the specified files from one disc to another.

- *DELETE <fsp>
  Minimum abbreviation : *DE.

Deletes the specified file.

- *DESTROY <afsp>
  Minimum abbreviation : *DES.

Deletes all the specified files from the disc in one action. A list of files to be deleted is first displayed and the user is asked to confirm the action.

- *DIR (<dir>)
  Minimum abbreviation : *DI

Changes the directory that is currently set up.

- *DRIVE (<drv>)
  Minimum abbreviation : *DR.

Changes the drive number to be accessed.

- *DUMP <fsp>
  Minimum abbreviation : *DU.

Produces a listing in hexadecimal values of the file specified.

- *ENABLE
  Minimum abbreviation : *EN.

Enables the use of the *BACKUP command.

- *EXEC <fsp>
  Minimum abbreviation : *E.

Executes the commands on the file.

- *HELP (<keyword>)
  Minimum abbreviation : *H.

Displays the commands and further information related to the sideways ROM chip identified by the <keyword>. The <keyword> for the disc filing system is DFS.

- *INFO <afsp>
  Minimum abbreviation : *I.

Displays detailed information for each of the specified files.

- *LIB <drv>
  Minimum abbreviation : *LIB

Sets the directory to be used as the library.

- *LIST <fsp>
  Minimum abbreviation : *LIST

Displays a text file on the screen with line numbers.

- *LOAD <fsp> (<address>)
  Minimum abbreviation : *L.

Reads the specified file into memory starting either at the specified address or the file's load address.

- *OPT 1 (n)
  Minimum abbreviation : *O.1

Enables or disables screen messages relating to any disc access.

- *OPT 4 (n)
  Minimum abbreviation : *0.4

Sets up the auto-start option of a disc.

- *RENAME <old fsp> <new fsp>
  Minimum abbreviation : *RE.

Changes the name of a file from <old fsp> to <new fsp>.

- *RUN <fsp> (parameters)
  Minimum abbreviation : *R.

Loads and runs a machine code program.

- *SAVE <fsp> <start address> <end address> (<execute address>) (<reload address>)
  Minimum abbreviation : *S.

Saves the specified section of memory onto the disc.

- *SPOOL <fsp>
  Minimum abbreviation: *SP.

Copies all text displayed on the screen onto the specified file until a further *SPOOL command is entered.

- *TITLE <name>
  Minimum abbreviation : *TI.

Change the name of the disc.

- *TYPE <fsp>
  Minimum abbreviation : *TY.

Displays a text file on the screen without line numbers.

- *WIPE <afsp>
  Minimum abbreviation : *W.

Deletes the specified files from the disc after asking for confirmation for each file in turn.

14. Error codes

This section lists the various error numbers returned in the basic variable ERR when an error is encountered as a result of a disc filing system command.

Error Number	Description
&BD	*Not enabled*
&BE	*Catalogue is full*
&BF	*Cannot extend file*
&C0	*Too many files open (maximum is 5);*
&C1	*Attempt to write to a file opened for read only*
&C2	*File already open*
&C3	*File locked*
&C4	*Attempt to rename a file with an existing name*
&C5	*Drive fault*
&C6	*Disc full*
&C7	*Disc fault*
&C8	*Disc changed while files still open*
&C9	*Disc is write protected*
&CA	*Bad checksum and possible data corruption*
&CB	*Bad number in *OPT command*
&CC	*Invalid file name*
&CD	*Invalid drive number*
&CE	*Invalid directory name*
&CF	*Invalid attribute following *ACCESS command*
&D6	*File not found*
&FE	*Command not found*

Further to the above errors which are produced by the disc filing system, there are the errors which originate from the disc controller chip. These are the following error numbers which are returned from any of the assembler disc access routines. You may also obtain these error numbers on the screen in the following form:-

DISK FAULT <error number> at <track address>

The error numbers are as follows:-

&08	*Clock error*
&0A	*Late DMA*
&0C	*ID CRC error*
&0E	*Data CRC error*
&10	*Drive not ready*
&12	*Write protect*
&14	*Track 0 not found*
&16	*Write fault*
&18	*Sector not found*

Appendix A: *Fitting the disc drive(s)*

This appendix gives technical information about setting up disc drives. The drives from VIGLEN have already been set up for you and you should not remove the disc unit from its case as this may invalidate your guarantee. Different makes of drives may have different names for the various links described in this appendix and thus the following description should only be treated as a general guide.

Drive Addressing

Drive addressing was introduced in *Chapter 7*. The data going to and from the drives goes down one cable therefore the drives must *'know'* which address they are so that they can ignore another drive's data. Every drive has what is known as a Drive Select Switchblock (DSS) inside the drive case on which options are to be selected, including which address the drive answers to. The DSS may be a set of links, plug-in links or a DIL (Dual In-Line) switch, every type having the links labelled as follows:-

HS
DSO
S1
DS2
MX
DS3
HM

If you have more than one drive then there are two modes of operation for them which are known as normal and fast access. This option is set with these links. In practice, discs are usually used in bursts with long periods of disuse in between. When the discs are not in use then their motors are off and their heads, just like tape recorder heads, are withdrawn from the disc. Before the disc can be read the motor must start up and spin the disc at the right rate, the head must then move to the disc and settle to exactly the right place.

All this takes time, but it couldn't be left going continuously, as the disc would wear out very quickly. As a compromise the disc drives assume that it is quite likely that disc operations will come in bunches, for example copying a file from one disc to another. If one drive is acessed then all the drive motors start up and are not turned off again until a few seconds after the last disc access. This means, that for each set of disc operations, the initial delay occurs only once.

The above paragraph did not mention the heads because you have the choice of whether the head is brought against the disc whenever the motors are on, which results in faster access in multiple disc operations, or whether the head is only brought against the disc when the particular drive is accessed, which will result in a longer life for the discs in drives not actually being used. To most people the extra fraction of a second taken to load the head will be quite insignificant, although a clicking noise will be heard, and they will choose the normal option. If you only have a single drive then the drive will be selected when the motor is on anyway making the choice of option irrelevant.

The links are made as follows:-

HS:	*made only if the normal option is required*
HM:	*made only if the fast access option is required*
DSO:	*made if the drive is to be drive 0*
DS1:	*made if the drive is to be drive 1*
DS2:	*do not make this link*
DS3:	*do not make this link*

Make sure that only one link out of HS and HM are made or the drive may behave peculiarly. Be especially careful that only one drive is drive 0 and that only one drive is drive 1. If two drives have the same address then they will '*argue*' with one another over the bus, that is the ribbon cable, with certain loss of data and possible damage to the drive circuitry. DS2 and DS3 are for other computers which don't access the second sides of 0 and 1 as 2 and 3.

Resistor pack

The data transfer rate along the ribbon cable is very high. At high frequencies electrical signals can actually bounce up and down the cable like echoes which causes errors as the original signal gets mixed up with the reflections. To prevent this the cable is '*terminated*' by a resistor pack at the very end. This resistor pack appears similar to an IC package and will be on the board of the drive itself. It contains one resistor for each of the lines in the cable. If you only have one disc drive then you do not need to do anything as this will already have been fitted. However, if you have two single drives then each one will have come with a resistor pack. As only one pack must be connected, the one which is not actually at the end of the cable must be removed. You will have a cable which goes from the computer to the first drive and then on to the second. The pack on the first must be removed.

Switching on

If the disc motors are on and the red indicator on the drives comes on when you switch on your drives for the first time then it usually means that the ribbon cable has become reversed. If only one drive comes on then reverse the cable on that drive. If both come on then either reverse both or reverse the cable at the computer end, checking to see whether the cable is coming out downwards.

If everything appears normal then try the drive addressing by typing '*CAT'. The lamp should come on on drive 0 and the motors should turn. If there is no disc in drive 0 then it will just sit there waiting. Press the <escape> key and, if you have a drive 1, type '*CAT 1'. The same should happen with drive 1.

Appendix B: *Care of discs*

Your discs will give you years of service if you follow some simple rules.

(i) Never turn the disc drives on or off with a disc inside. This will not damage the disc physically but some of the data may be lost and you may have to format the disc again before you can use it. If you have a power cut then this will be beyond your control so it is a good idea to keep backups of your most valuable programs.

(ii) Data on the disc may be erased by proximity to transformers in electrical equipment. If the transformer is shielded, as the one in the drive itself has to be, then it is usually okay. What you have to be careful of are small charger transformers for calculators and other pieces of equipment which only have unshielded plastic cases.

(iii) Do not let the discs get hot because this will also erase the data. They must not be left in the sun, on a radiator or near anything hot like a television.

(iv) Do your best to keep the discs dust free. Keep the access doors to the drives closed and always keep the discs in their protective sleeves. Never touch the surface of the disc and don't smoke whilst handling them.

(v) For 5¼ inch discs do not fold or roll discs up as they will not work if they get creased. Do not even write on the labels once they are struck to the disc unless you do it very lightly with a felt tipped pen.
Note: Berol sell a special pen which has a tip that bends if you press too hard.

(vi) For 3 inch discs do not attempt to open the shutter and touch the disc surface.

Appendix C: *Disc sector access*

The OSWORD call with A=&7F, or A%=&7F in BASIC, will allow you to read and write sectors independently of the DFS. The X register, or X% in BASIC, contains the low byte and the Y register, or Y% in BASIC, contains the high byte of the address of a parameter block in memory.

The block must be set up as follows:-

Offset from YX	**Contents**
0	*drive number*
1 - 4	*start address of Source or Destination data*
5	*number of parameters in command (n=3 for load/save)*
6	*command (load=&53, save=&4B)*
7+n	*parameters*
8+n	*space for result code*

For a read/write operation, the first parameter is the track address and the second parameter is the sector address on that track, which can be from 0 to 9. The third parameter is a combination of the number of sectors to be accessed in bits 0-4 and the length of the disc record coded in bits 5-7. This will have the value &21 on the BBC computer. On exit, the result code will indicate whether the operation was successful or not. A zero in this byte indicates a successful operation. If this byte is not zero then it will contain an error code.

If the error code is any of the following then it is worth attempting the operation again:-

(i)	8	*clock error*
(ii)	10	*late DMA*
(iii)	12	*ID CRC error*
(iv)	14	*data CRC error*

Bit 0 of the error code is always zero. The error should be considered unrecoverable after 10 retries.

Appendix D: *Contents of sectors 000*

The procedure given in *Appendix 3* is used to load sectors 000 and 001 then the catalogue information can be directly examined. This appendix describes the format of this information within the sectors. Addresses are usually 16 bits long because this is the addressing range for the 6502 CPU. However the disc system is compatible with the tube hence the addresses are extended to 18 bits. If the extra two bits are both set to one then the address is assumed to be in the main computer. The two extra bits will be referred to as the high order bits, the next eight bits will be referred to as the middle byte and the last eight bits will be referred to as the low byte. The start sector of a file is 10 bits long and consists of two high order bits and a low order byte.

The format of the sectors is as follows:-

SECTOR 000

Bytes	**Meaning**
&00 to &07	*disc title (first 8 characters)*
&08 to &0E	*first filename*
&0F	*directory letter of first file*
&10 to &16	*second filename*
&17	*directory letter of second file*
"	
"	
"	

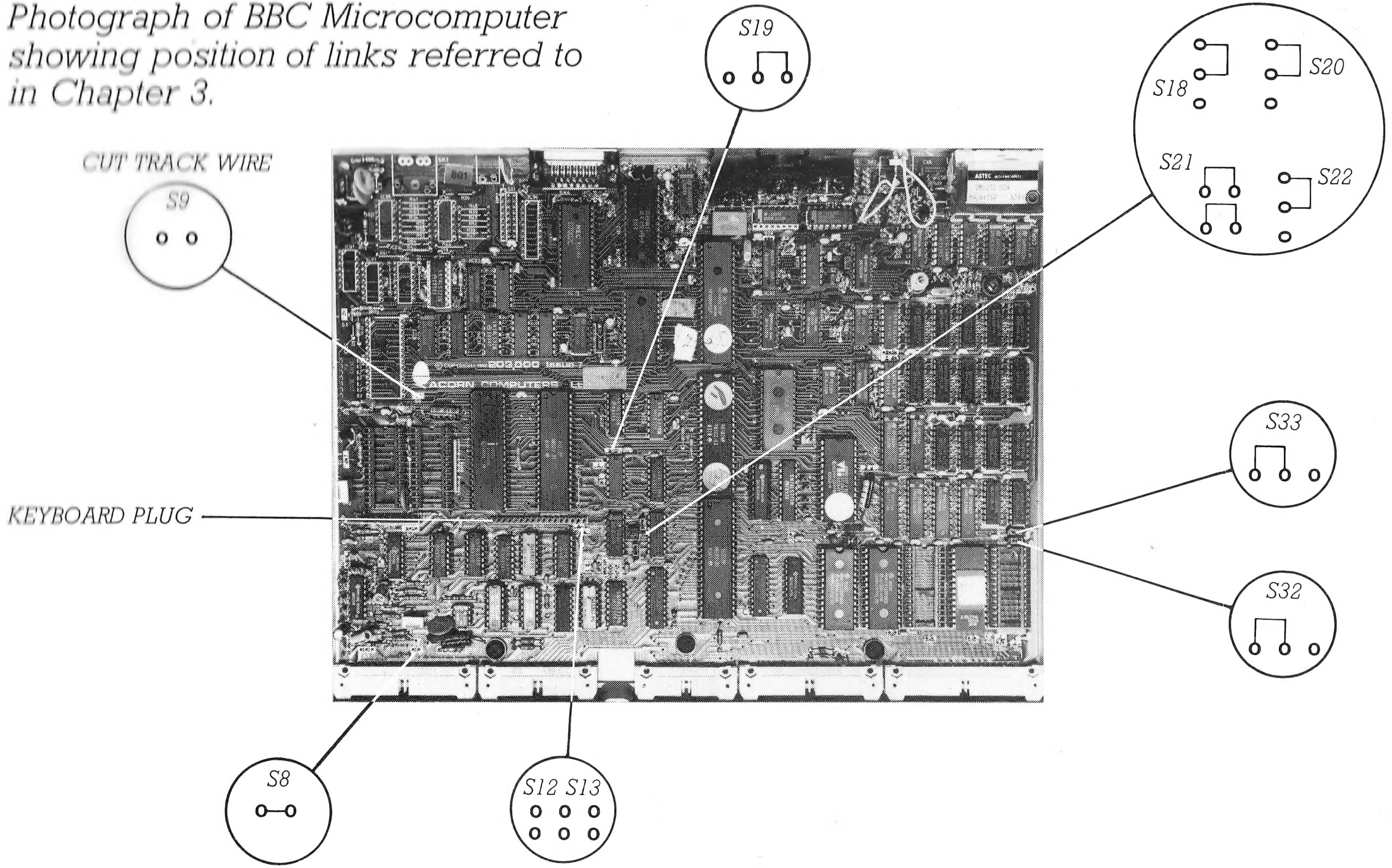
repeated for up to 31 files

SECTOR 001

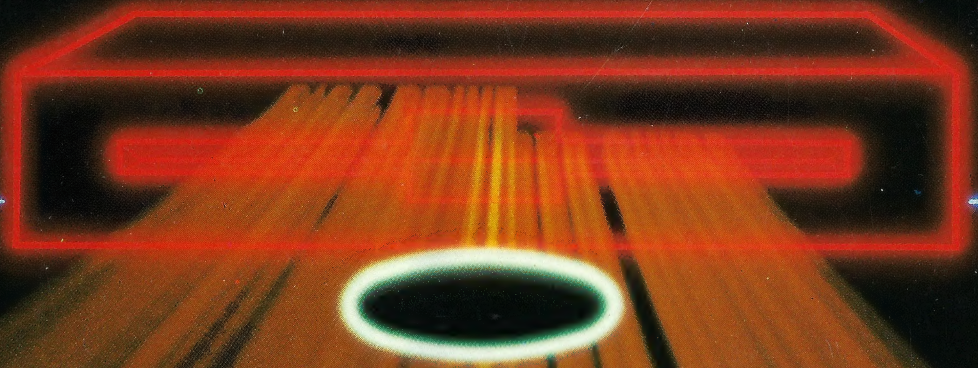
Bytes	**Meaning**
&00 to &03	*last four characters of title*
&04	*issue number of disc*
&05	*eight times the number of catalogue entries*
&06 bits 0,1	*number of sectors on disc high order bits*
&06 bits 4,5	*auto boot option*
&07	*number of sectors on disc low order byte*
&08, &09	*load address of first file middle and low byte*
&0A, &)b	*execution address of first file middle and low byte*
&0C, &OD	*length of first file middle and low byte*
&0E bits 0,1	*start sector of first file high order bits*
&0E bits 2,3	*load address of first file high order bits*
&0E bits 4,5	*length of first file high order bits*
&0E bits 6,7	*execution address of first file low order bits*
&0F	*start sector of first file low order byte*
"	
"	
"	

repeated for up to 31 files

Photograph of BBC Microcomputer showing position of links referred to in Chapter 3.



£4.95
inc.VAT



DISC DRIVE USER GUIDE

FOR THE BBC MICROCOMPUTER